PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 01/06

Towards a Fault-Tolerant Open Multi-Agent Platform based on a Law-Governed Approach

Maíra Athanázio de Cerqueira Gatti
Carlos José Pereira de Lucena
Jean-Pierre Briot
Zahia Guessoum

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 01/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December, 2005

Towards a Fault-Tolerant Open Multi-Agent Platform based on a Law-Governed Approach *

Maíra Athanázio de Cerqueira Gatti[1], Carlos José Pereira de Lucena[1], Jean-Pierre Briot[2] and Zahia Guessoum[2]

[1]Laboratório de Engenharia de Software – LES
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil
{mgatti,lucena}@inf.puc-rio.br

[2]LIP6, Universit´e Pierre et Marie Curie (Paris 6)
8 rue du Capitaine Scott, 75015 Paris, France
{Jean-Pierre.Briot, Zahia.Guessoum}@lip6.fr

Abstract. Multi-agent systems are composed by autonomous components (agents) which interact to achieve their goals. Due to their autonomy and no control point of decisions making, it is hard to control and forecast its behavior during their interactions. If we consider that fault tolerance is a technique which can be very useful to increase the reliability of a distributed system, and considering that an Open Multi-agent System (MAS) is a special type of such systems, there is a need to study the particular characteristics of these Open MAS in order to reach higher degrees of dependability. Despite all the research done in the last years on the development of fault-tolerant applications and of approaches to enforce secure interaction protocols in multi-agent systems, there is no mechanism which uses the law elements of interaction's control which improves the agent criticality analysis. As this mechanism would increase dependability of those systems, it is proposed a solution to achieve it through the integration of the DarX framework, which is a framework for dynamic replication, and the XMLaw, which is a declarative language for implementing the law enforcement approach for regulating agents' interaction.

Keywords: Multi-agent systems, distributed systems, open systems, dependability, availability, criticality, reliability.

Resumo. Sistemas multi-agentes são sistemas compostos de componentes de software (agentes) autônomos que interagem para alcançar seus objetivos. Devido à autonomia e descentralização da tomada de decisões, torna-se difícil o controle e a previsão do comportamento resultante da interação entre esses agentes. Se considerarmos que tolerância a falhas é uma técnica que pode ser muito útil para aumentar a confiabilidade de um sistema distribuído, e considerando que Sistema Multi-Agente Aberto (SMA) é um tipo deste sistema, verifica-se a necessidade de um estudo das características particulares destes sistemas abertos para alcançar um elevado grau de fidedignidade. Apesar de toda a pesquisa feita nos últimos anos no desenvolvimento de aplicações tolerantes a falhas, e as abordagens para garantir interações seguras de protocolos em sistemas multi-agentes, não existe nenhum mecanismo que utilize elementos das leis de controle de interação que auxiliem no cálculo da criticalidade dos agentes. Como esse mecanis-

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil

mo aumentaria a fidedignidade desses sistemas, propomos uma solução para alcançar este objetivo através da integração do framework Darx, que é um framework para replicação dinâmica, e o XMLaw, que é uma linguagem declarativa para implementar a abordagem de leis para regular as interações de agentes.

Palavras-chave: Sistemas multi-agentes, sistemas distribuídos, sistemas abertos, fidedignidade, disponibilidade, criticalidade, confiabilidade.

In charge for publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Table of Contents

1 Introduction

Multi-Agent systems (MAS) are decentralized and self-organizing systems that consist of autonomous entities (agents) that solve tasks by cooperation [1]. The agents coordinate their actions in a decentralized manner by sending and receiving messages. The absence of centralized coordination data makes it hard to determine the current state of the systems and/or to predict the effects of actions. Moreover, agents are therefore faced with significant degrees of uncertainty for making decisions since it is very hard to devise all the possible situations that may arise in the execution context. Taming this uncertainty is a key issue for open software development. In such circumstances, reliability is a strategy for dealing with uncertainty associated with interactions in open systems [2].

However, in critical applications such as business environments or government agencies (hospitals, police, justice, etc.), the behavior of global system must be taken into account and structural characteristics of the domain have to be incorporated [3]. Concepts as organizational rules [4], norms and institutions [5][6], and social structures [7] arise from the idea that the effective engineering of MAS needs high-level, agent-independent concepts and abstractions that explicitly define the organization in which agents live [7]. These are the rules and global objectives that govern the activity of an enterprise, group, organization or nation. From the organizational point of view this creates a need to check conformance of the actual behavior of the society to the behavior desired by the organization [8]. Hence, these laws must be enforced to delimit tolerated autonomous behavior and are also used to foster the development of trusted systems. The bottom line is that laws are used to represent the valid interactions in open MAS applications [2].

Due to the need of enforcing the laws through the agents' interaction, some works have appeared in the last years to propose mechanisms for regulating agent's interactions in complex scenarios. Here we are going to use the approach proposed by [9] t-hrough XMLaw description language, which allows for the runtime enforcement of agents compliance based on a law-governed mechanism.

Moreover, the dependability of a computing system is its ability to deliver service that can justifiably be trusted. Correct service is delivered when the service implements the system function. Considering that it is very hard to determine the state of autonomous systems due to the absence of centralized coordination data [11], there is the need to ascertain open multi-agent systems dependability, that is a justified trust that they will satisfactory perform their missions and not cause any catastrophes [10].

Fault tolerance is said to be a useful technique to increase the reliability of a distributed system. Considering that an Open Multi-agent System (MAS) is a special type of such systems, there is a need to study the particular characteristics of these Open MAS in order to reach higher degrees of dependability.

The Agent Replication Technique, which is the act of creating one or more duplicates of one or more agents in a multi-agent system, seems to be an efficient way to achieve fault tolerance in an Open Multi-agent System, since fault tolerance is generally implemented by error detection and subsequent system recovery. Hence we can increase the dependability of an Open Multi-agent System just by implementing the Agent Replication technique. It is also necessary that this approach should be dynamic and adaptative since this kind of Open MAS are very dynamics [12]. In this work, we

choose to use the DarX framework, which is a framework for dynamic replication of agents that ensure dependability as transparent as possible to the application. It focuses on scalability and adaptive for large-scale multi-agent systems.

If our main goal is to arise as much as possible the dependability of multi-agent systems in order to guarantee a high degree of reliability of that kind of systems, a platform that integrate these two approaches (dependability achieved by fault tolerance and by governance of agents' interaction) would be an efficient way of doing it. So, we can have a platform with high degree of dependability and a law-governed approach to improve the criticality analysis which defines the number of agent's replicas and at the same time we could provide the ability of monitoring and enforcing the behavior of agents through the enforcement of laws.

We will briefly describe in the second section the basic concepts of dependability in computing systems, and the associated notion and mechanisms of fault tolerance. It is also presented an overview of DarX, which is a framework for dynamic replication, and its mechanisms that we are using. The third section introduces XMLaw, which is a declarative language for implementing the law enforcement approach for regulating agents' interaction in an open multi-agent system that we are also using. The fourth section presents the problem description within the problems to be tackled illustrated by a scene of negotiation. Finally we present the expected contributions in the fifth section and after that, in the sixth section, it is detailed the proposed solution to achieve these contributions.

2 Dependability and Fault Tolerance

Dependability has been for a long time a major concern in ubiquitous computing systems which control structures as critical as railroads, planes, and nuclear plants. So, concepts and techniques are well established [10]. The concepts described in this section are based on [14] and [15].

There are two concepts related to dependability. The main one says that the dependability of a computing system is its ability to deliver service that can justifiably be trusted. Correct service is delivered when the service implements the system function, i.e., what the system is intended to do. The second one says that dependability is the ability to avoid service failures that are more frequent or more severe than is acceptable. Considering that a service failure is an event that occurs when the delivered service deviates from correct service, when service failures are more frequent or more severe then acceptable then we have a dependability failure.

Considering the main notion of dependability, three concepts further describe this notion: the attributes of, the threats to, and the means by it is attained (Figure 1) [14].

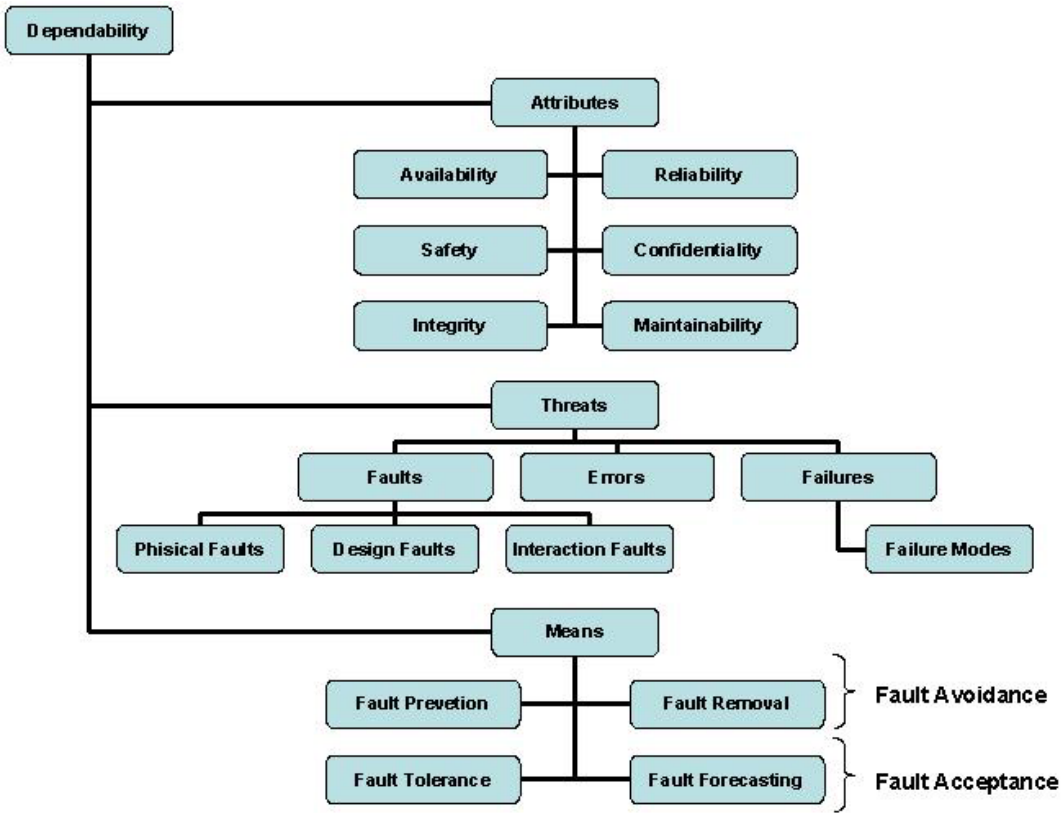

Figure 1 - Dependability tree

The attributes of system dependability consist of: (i) availability, the deliverance of correct service at a given time; (ii) reliability, the continuous deliverance of correct service for a period of time; (iii) safety, the absence of catastrophic consequences on the users and the environment; (iv) confidentiality, the absence of unauthorized disclosure of information; (v) integrity, the absence of improper system state alterations; (vi) maintainability, the ability to undergo repairs and modifications.

Dependability is an integrative concept that encompasses these basic attributes; depending on the application intended for the system, different emphasis may be put on each attribute. Several other dependability attributes have been defined that are either combinations or specialization of the above [10].

The threats to a system's dependability consist of failures, errors and faults. The ways in which a system can fail are its failures modes, characterized by the severity and the symptoms of a failure. A fault is active when it produces an error; otherwise it is dormant.

The means to attain a system's dependability were regrouped in four techniques: fault prevention, fault removal, fault tolerance and fault forecasting. However, our focus will be in the fault tolerance, i.e., how to deliver correct service in the presence of active faults. It is generally implemented by error detection and subsequent system recovery, and possibly by error containment. Recovery transforms a system state that contains one or more errors (and possibly faults) into a state that can be activated again without detected errors and faults.

There are three techniques applicable for fault tolerance: (i) tolerance of physical faults; (ii) tolerance of design faults; (iii) tolerance of interaction faults. We are going to address the first and the last one since it solves both problems of accidental interaction faults, which in our context could be an agent in deadlock, and/or intentionally malicious interaction faults, that could be a harmer agent intended to destroy the system or another agent interacting on it.

Several approaches ([17], [18], [19], [20], [21]) address the multi-faced problem of fault tolerance in multi-agent systems. These approaches can be classified in two main categories. The first category focuses especially on the reliability of an agent within a multi-agent system. This approach handles the serious problems of communication, interaction and coordination of agents with the other agents of the system [12]. The second category addresses the difficulties of making reliable mobile agents which are more exposed to security problems [16]. This second category is beyond the scope of this work. The main limit of current replication techniques is that most of them are not quite suitable for implementing adaptive replication mechanisms [12].

Considering that limitation, a specific and novel framework for replication, named DarX (see details en next section), which allows dynamic replication and dynamic adaptation of the replication policy (e.g., passive to active, changing the number of replicas) was designed. It was designed to easily integrate various agent architectures, and the mechanisms that ensure dependability are kept as transparent as possible to the application.

2.1 DarX: A Framework for Dynamic Replication

Replication mechanisms have been successfully applied to various distributed applications, e.g. data-bases. But in most cases, replication is decided by the programmer and applied statically, before the application starts. This works fine because the criticality of components (e.g., main servers) may be well identified and remains stable during the application session. Opposite to that, in the case of dynamic and adaptive multi-agent applications, the criticality of agents may evolve dynamically during the course of computation. Moreover, the available resources are often limited. Thus, simultaneous replication of all the components of a large-scale system is not feasible [13].

There are several approaches to fault tolerance in MASs that can be classified in two groups to know: agent-centric approaches, which build fault tolerance into the agents; and system-centric approaches, which move the monitoring and fault recovering into a separate software entity. Agent replication uses aspects of both agent-centric and system-centric approaches. Each agent must have the capability to utilize replication [22].

The agent replication technique is already old to the people who work with fault tolerance in distributed systems. Then the agent replication would be the act of creating one or more replicas of one or more agents. The replication degree, or number of each agent replica, will be the criticality degree of it and the how much complex it is to add or remove members in the existent group. There are two main types of replication protocol: active and passive protocols. The first one occurs when all replicas process concurrently all input messages. And the second one occurs when only one of the replicas processes all input messages and periodically transmits its current state to the other replicas in order to maintain consistency [12].

DarX will enable to switch to the most suitable dependability protocol. It includes group membership management to dynamically add or remove replicas. It also provides atomic and ordered multi-cast for the replication groups' internal communica-

tion. Messages between agents, that is communication external to the group are also logged by each replica, and sequences of messages can be re-emitted for recovery purposes.

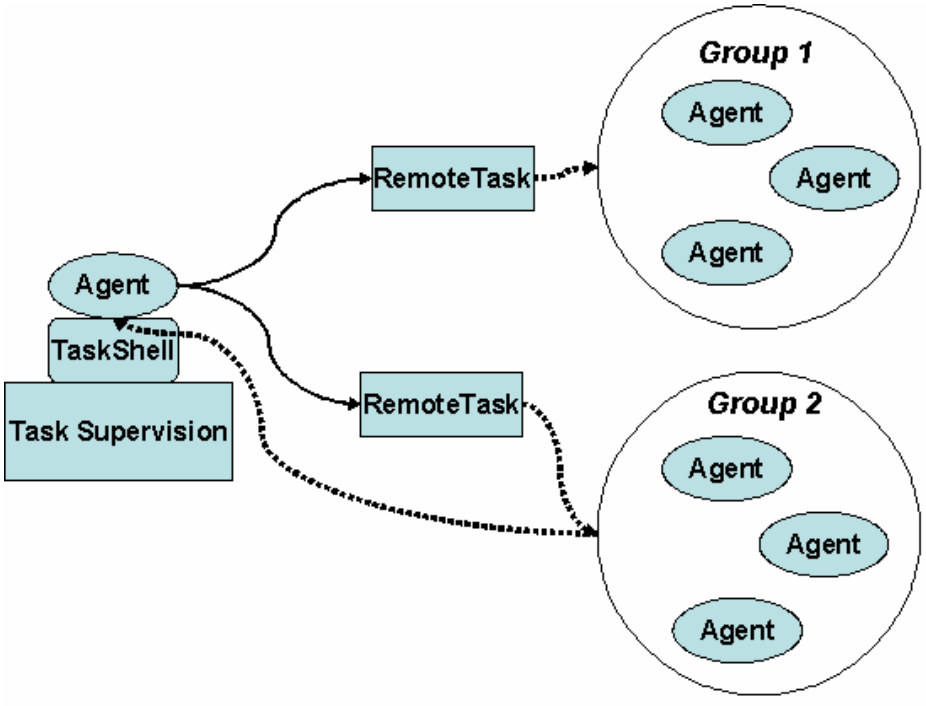

Figure 2 - DarX application architecture

DarX also provides global naming. Each agent has a global name which is independent of the current location of its replicas. The underlying system allows handling the agent's execution and communication. Each agent is itself wrapped into a TaskShell (Figure 2), which acts as a replication group manager and is responsible for delivering received messages to all the members of the replication group, thus preserving the transparency for the supported application. Input messages are intercepted by the TaskShell, enabling message caching. Hence all messages get to be processed in the same order within a replication group.

An agent can communicate with a remote agent, regarding whether it is a single agent or a replication group, by using a local proxy implemented by the RemoteTask interface. Each RemoteTask references a distinct remote entity considered as its replication group leader. The reliability features are thus brought to agents by an instance of a DarX server (DarxServer) running on every location. Each DarxServer implements the required replication services, backed up by a common global naming/location service.

DarX provides the needed adaptive mechanisms to replicate agents and to modify the replication strategy. Meanwhile, we cannot always replicate all the agents of the system because the available resources are usually limited. The problem therefore is to determine the most critical agents and then the needed number of replicas of these agents. The resources are thus allocated to critical agents. Then there are two cases that might be distinguished: 1) the agent's criticality is static and 2) the agent's criticality is dynamic. In the first case, multi-agent systems have often static organization structures, static behaviors of agents, and a small number of agents. Critical agents can be

therefore identified by the designer and can be replicated by the programmer before run time.

In the second case, multi-agent systems may have dynamic organization structures, dynamic behaviors of agents, and a large number of agents. So, the agents criticality cannot be determined before run time. The agent criticality can be therefore based on these dynamic organizational structures. The problem is how to determine dynamically these structures to evaluate the agent criticality? [12] proposed a way of determining it through role analysis. In this case, there are two approaches: (i) some prior input from the designer of the application is needed for specifying the roles' weights; and (ii) there is an observation module to each DarxServer that collect the data through the agent execution and their interactions. In the second approach, global information is built and then used to obtain roles and degree of activity to compute the agent criticality.

Another way of determining dynamically these structures to evaluate the agent criticality was proposed in [23]. In that case, a multi-agent system is therefore represented by a graph which reflects an emergent organizational structure. This structure can be interpreted to define each agent criticality. The hypothesis is that the criticality of an agent relies on the interdependences of other agents on this agent. The interdependence graph is initialized by the designer, and then it is then dynamically adapted by the system itself. They proposed some algorithms to dynamically adapt it and describe them in [23].

In the fourth section we will highlight the main questions about determining dynamically these structures to evaluate the agent criticality considering the law-enforcement approach. Then, in the fifth section, we describe the contributions of doing it to finally, in the sixth section, detail the proposed solution.

3 Law-Governed Interaction

As we have already seen, distributed software agents are independently implemented, i.e., the development takes place without a centralized control. In order to achieve a coherent system, we assume that every agent developer may have an a priori access to the open system specification, including the protocol description and the interaction laws. In this scenario, law-governed architectures can be designed to guarantee that the specifications will be obeyed.

In this kind of architecture, a mediator is needed to intercept messages and interpret the laws that were previously described. In fact, to provide a scalable solution, this approach needs to consider the existence of a pool of mediators. As more clients interact within the system, additional mediator's instances can be added to improve throughput. The core of a law-governed approach is the mechanism used by the mediators to monitor the conversations between components. A software support was developed [24] that when necessary permits the extending of this basic infrastructure to fulfill any open system requirements or interoperability concerns.

Besides monitoring the conversation of agents, it is necessary to specify the interaction rules that will govern this mediation. In this section, we explain the description language XMLaw [9]. XMLaw is used to represent the interaction rules of an open system specification. Those rules are interpreted by a mechanism that at runtime is used to analyze the compliance of open MAS with interaction laws

3.1 XMLaw

The model presented in this section is used to represent interactions in an open distributed environment. Basically, interactions should be analyzed, and after that, described using the concepts proposed in the model. Then, the concepts are mapped to a declarative language, called XMLaw.

Interaction's definitions are interpreted by a software framework that monitors components' interaction and enforces the behavior specified on the language. Once interaction is specified and enforced, despite the autonomy of the agents, the system's global behavior is better controlled and predicted. Interaction specification of a system is also called laws of a system. This is because besides the idea of specification itself, interactions are monitored and enforced. Then, they act as laws in the sense that describe what can be done, what cannot be done and what should be done. Bellow, it is presented each concept and their related representation in XMLaw.

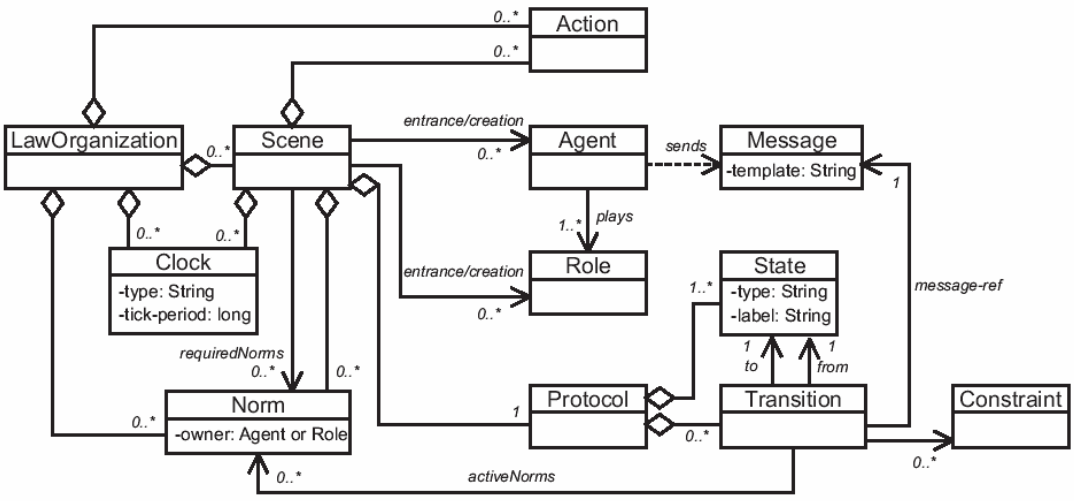

Figure 3 - Conceptual Model of Interaction

This model provides computational concepts that allow specify interaction laws in multi-agent systems. Most of this model was already reported in [25] and therefore the concepts are very briefly introduced in this paper to make it self-contained.

The outer concept of this model is the LawOrganization. We can see this element as representing the interaction laws (or normative dimension) of a multi-agent organization. A LawOrganization is composed of scenes, clocks, norms and actions. Scenes are interaction contexts that can happen in an organization. They allow modularizing interaction breaking the interaction of the whole system in smaller parts. Clocks introduce global times which are shared by all scenes.

Norms capture notions of permissions, obligations and prohibitions regarding agents' interaction behavior. Actions can be viewed as a consequence of any interaction condition, for example, if an agent acquires an obligation, then the action 'A' should be executed.

Scenes define an interaction protocol (from a global point of view), a set of norms and clocks that are only valid in the context of the scene. Furthermore, scenes also identify which agents are allowed to start or participate of the scene. Non-deterministic

automatons, enhanced by the constraint element and by the possibility of integration with the other elements from the model, represent interaction protocols. Constraints are used as conditional firing of protocol transitions allowing to check complex conditions based on the message contents.

Figure 3 can be viewed as a structural and static view from the concepts used to specify interactions. These elements relate to each other through the associations, dependencies and aggregations relationships showed in this figure. However, to achieve lower coupling levels among these elements, there is also a dynamic model based on events that reify the relationships of the conceptual model in a flexible way.

Events are the basis of the communication among law elements, that is, law elements dynamically relate with other elements through event notifications. Basically, we can understand the dynamic of the elements as a chain of causes and consequences, where an event can activate a law element; this law element could generate other events and so on. For example, the arrival of a message generates an event (message arrival), this event may activate a transition (transition activation), transition in its turn may activate a clock (clock activation), which generates a (clock tick) event, and lastly it activates a norm (norm activation). Figure 4 shows this chain of causes and consequences, and Figure 5 summarizes all law elements that can generate and sense events.

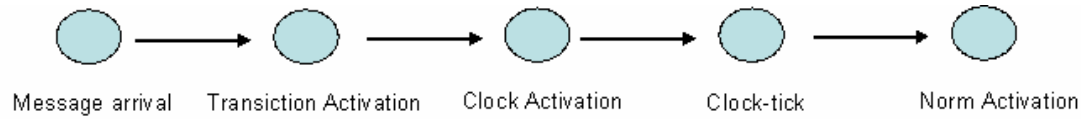

Figure 4 - Chain of Events

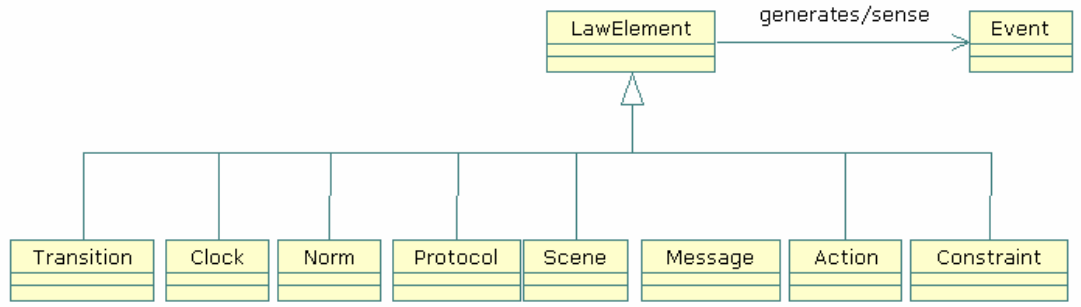

Figure 5 - Law Elements that generates/senses events

There is a framework that supports development of open distributed systems providing compliance with both the model of interactions proposed previously and the declarative language XMLaw. The framework has a set of modules that supports three types of users: (i) "Law developer" represents the developer responsible for specifying the laws, he must understand the application under construction, know the law concepts, and then, specify the laws for the application; (ii) "Agent developer" represents the developer responsible for building the agents of a multi-agent system, those developers know about the existence of the laws and should design the agents in compliance with them; (iii) "Software infrastructure developer" deals with law enforcement software support. Sometimes there is no software infrastructure that implements the law

enforcement mechanism or the existent infrastructures are not suitable for the systems that are under development. In these cases, software infrastructure developers should modify the existent infrastructure, or even construct a new one.

The framework provides support for each one of those developers. First, it is provided a declarative language, the XMLaw, for law developers, which allows the specification and maintenance of the interactions of multi-agent systems. XMLaw specifications are interpreted and enforced by the law enforcement framework. The framework provides this software support, which contains a number of hotspots that can be extended by software infrastructure developers. Lastly, agent developers are provided with classes and interfaces that support building agents integrated to the law enforcement software.

Most of the framework is implemented as mediator agent modules. The mediator agent monitors all interactions and makes sure that interactions are compliant with the specifications. The mediator performs a number of activities. First, the mediator waits for receiving messages. Once a message arrived, it checks if the message belongs to the mediator protocol. If it does, the mediator proceeds with the protocol execution. Otherwise, if the message belongs to some agent conversation, the mediator starts the process of enforcing, and if the laws allow, the message is redirected to the addressee agent. This sequence of activities is repeated while the mediator agent is running.

The communication among the modules is mainly based on event notifications. This approach leads to a low coupling level among modules and also leads to more flexible system designs [26]. In event-based systems, there is a module, named Event Manager, which provides an interface containing all operations needed for enabling event-based communication.

The proposal here is not to detail the framework, so more details can be found in [27]. Next sections will address both DarX and XMLaw, so it is very important that all the concepts are restrained so a better exploitation will be done.

4 Problem Description

To illustrate the problem mentioned in section two when describing DarX about how to dynamically calculate the agent criticality considering the organizational structures, we will describe a negotiation scene based on FIPA-CONTRACT-NET [28], which is represented in Figure 6. The goal is do discover how the elements of the XMLaw could improve the agent criticality analysis that is done by DarX.

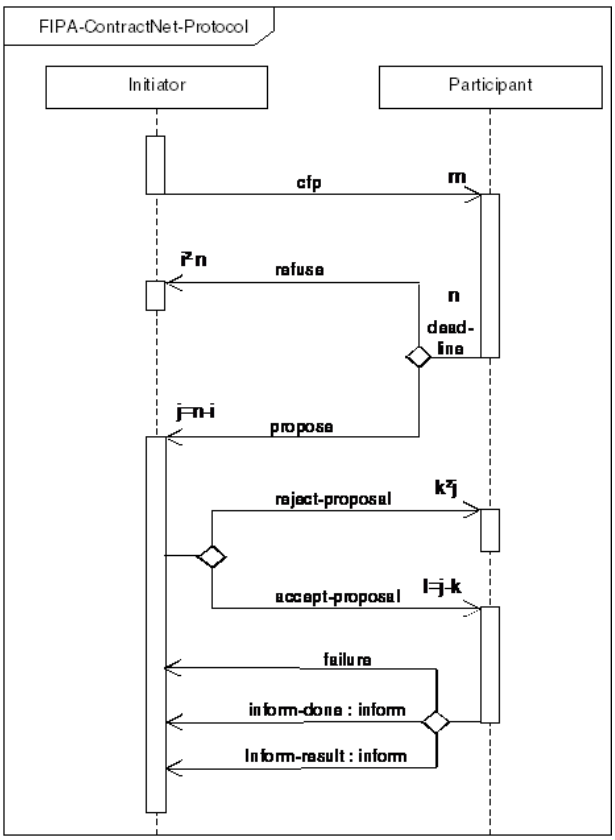

Figure 6 - FIPA CONTRACT-NET PROTOCOL

Basically, the Initiator solicits *m* proposals from other agents by issuing a call for proposals (`cfp`) act, which specifies the task. Participants receiving the call for proposals are viewed as potential contractors and are able to generate *n* responses. Of these, *j* are proposals to perform the task, specified as `propose` acts.

The Participant's proposal includes the preconditions that the Participant is setting out for the task, which may be the price, time when the task will be done, etc. Alternatively, the *i=n-j* Participants may `refuse` to propose. Once the deadline passes, the Initiator evaluates the received *j* proposals and selects agents to perform the task; one, several or no agents may be chosen. The *l* agents of the selected proposal(s) will be sent an `accept-proposal` act and the remaining *k* agents will receive a `reject-proposal` act. The proposals are binding on the Participant, so that once the Initiator accepts the proposal, the Participant acquires a commitment to perform the task. Once the Participant has completed the task, it sends a completion message to the Initiator in the form of an `inform-done` or a more explanatory version in the form of an `inform-result`. However, if the Participant fails to complete the task, a `failure` message is sent.

Now, suppose that this protocol was described as a state machine (Figure 7) where *Si* are the protocol's states during its execution and the clocks' representation are the clocks activation and deactivation for each + or -, respectively. The protocol starts with the state S0 when the Initiator solicits *m* proposals from other agents and it ends with the states S4, or S5, or S7, it depends on the protocol's flow.

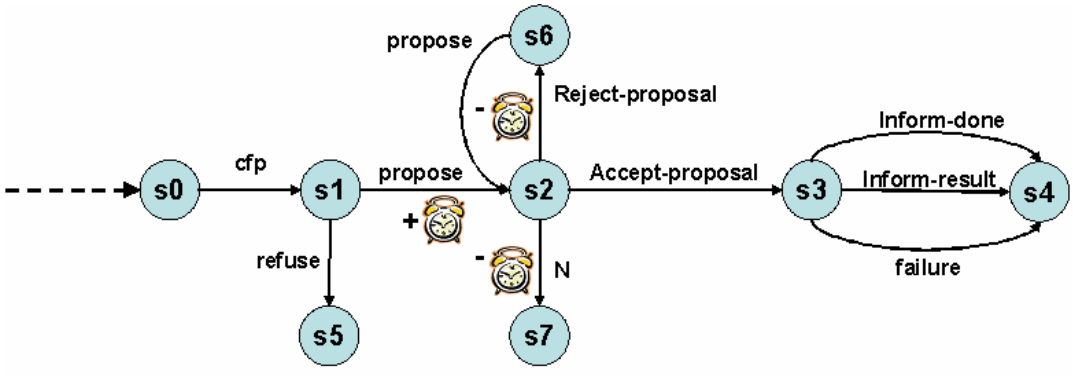

Figure 7 - Contract-Net State Machine

Now, considering this representation we are going to map another negotiation scene in other to check how the criticality of the agent that is executing the protocol vary. Then suppose the following situation: a customer starts a negotiation sending a proposal for a book to a seller. He informs the maximum price that he will pay for the book. The seller can accept with a proposal or can refuse it. If he accepts, he can send proposals with lesser or equal price informed by the customer. When the customer receives the proposal, he has 20 seconds to decide if he will accept it or not. After 20 seconds, if the customer hasn't answered the seller, he can sell the product to another customer. Otherwise the seller is not allowed to sell it to anybody else. If the customer refuses it, the seller can re-propose another price. If the customer accepts it, the seller informs the bank where the payment must be done. Then the customer has the obligation of paying for the product and of informing the number of the voucher to the seller. The scene ends then when the customer informs that paid it with the voucher number.

Figure 8 shows the state machines of the negotiation protocol and all XMLaw elements (clock and norm) activated during its execution.

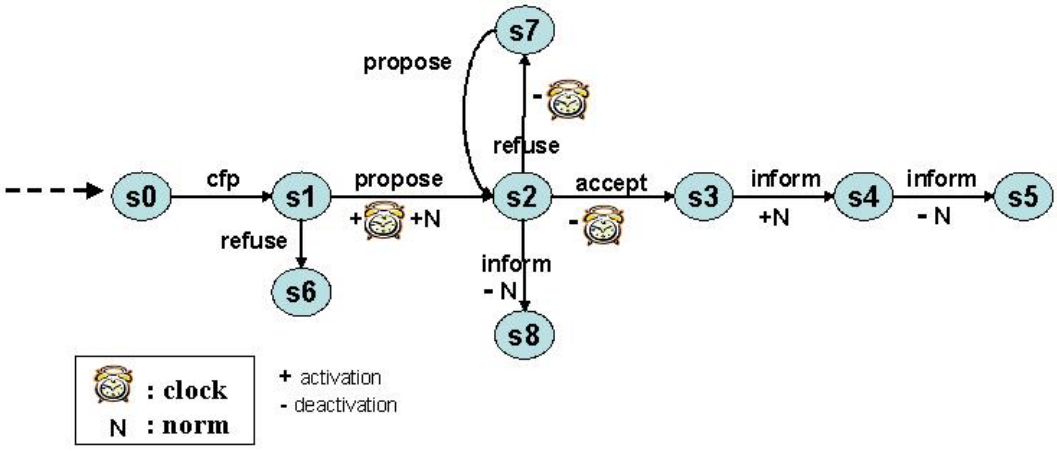

Figure 8 – Negotiation Scene Description

To summarize it, we have:

- From state S0 to S1: the customer starts a negotiation sending a proposal for a book to a seller (**cfp** message).
- From state S1 to S2: the seller accepts the customer's proposal. He sends proposals with lesser or equal price informed by the customer (**propose** message). This

transition activates the clock and the customer has 20 seconds to answer the seller.

- From state S1 to S6: the seller refuses the customer's proposal (**refuse** message) and the protocol ends.
- From state S2 to S3: the customer accepts the seller's proposal before the 20 seconds (**accept** message).
- From state S2 to S7: the customer refuses the seller's proposal before the 20 seconds (**refuse** message).
- From state S7 to S2: the seller proposes the customer again with another price (**propose** message) and the clock is activated again.
- From state S2 to S8: the customer doesn't answer the seller and the seller informs the customer that he can offer the book to another customer (**inform** message) and the protocols ends.
- From state S3 to S4: the seller informs the customer the bank where he has to pay for the book (**inform** message) and he has the obligation to pay in order to receive the book (norm activation).
- From state S4 to S5: the customer informs the seller the voucher (**inform** message) and has the permission the receive the book. The protocol ends.

If we consider that when an event (as clock activation/deactivation, norm activation/deactivation, etc.) occurs during the scene execution the agent critilicality can increase or decrease, since the agent becomes more or less important, each element should be analyzed in order to calculate it in the best way. Moreover, another elements and events that might not be handled by XMLaw should be analyzed in order to evaluate how it could influence the agent criticality analysis. For example, when an agent starts playing a role its criticality may increase or decrease.

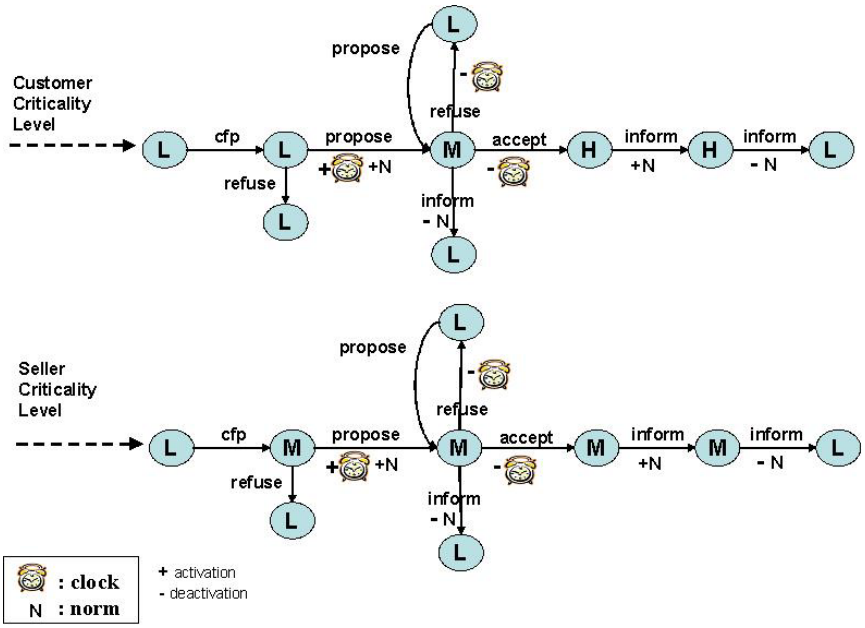

Figure 9 - Criticality Analysis

In the context of the negotiation scene, when the customer have to answer the seller if he will accept his proposal or refuse it, his criticality may increase, since that agent became very important to the seller and should not crush, for example. So, if we could classify the criticality of the agent in three levels, we would have a low (L), medium (M) or high (H) criticality level during the scene execution. Figure 9 shows how it could vary in a time period. From s1 to s2 states the customer criticality level increased from low to medium since the clock activation event was fired. From s2 to s8 or s7 states it decreased since the clock deactivation event was fired. From s2 to s3 states it increased even more going from medium to high since the customer has the obligation of paying for the product. Now, if we analyze the seller criticality during the scene execution, we can perceive that from s1 to s2 states its criticality level increased since the customer has called him for a proposal, then he has to answer him.

The main questions which arise from that analysis are:

- How and which elements of the XMLaw could improve the agent criticality analysis that is done by DarX?
- How can we do it in the best way considering coupling, modularity and reuse?

Next sections will describe the expected contributions as a consequence of answering those questions and how to solve it.

5 Expected Contributions

As we have already seen, the Open Multi-Agents System dependability can be achieved by fault tolerance. Among others existing fault tolerance techniques, there is a specific one which has been used in the recent years for achieving dependability in multi-agent systems. It is the Agent Replication technique. It has been used by [17], [18], [19], [20], [21] in several approaches and we are going to use it in our work from the point of view of [12][13][23], since it is an effective way to implement fault tolerance for distributed systems.

Furthermore, we are going to use XMLaw since it also increases Open Multi-Agents System reliability by law enforcement approach for regulating agents' interactions through a higher control.

This work presents an extension of the XMLaw conceptual model described in the third section as a way of improving its dependability. We propose to use new elements that help specifying the attributes concerning the agent criticality during its interaction with other agents.

Moreover, considering that XMLaw framework is an event-based framework, other elements from the law's specification that are perceived by events can improve the criticality analysis done by DarX which is used for calculating the agent number of replicas as clock activations, norms activations, etc.

We believe that by integrating DarX and XMLaw we can provide a fault-tolerant Open Multi-Agent Platform based on a Law-Governed Approach with high dependability, in particular considering availability and reliability.

6 Proposed Solution

In order to integrate the XMLaw and DarX it was necessary to extend both of them. First we analyzed XMLaw and studied which elements should be inserted, which events should be sensed by the new elements and so on. It was not a trivial work considering that XMLaw is not an extensible framework for adding new elements or generating new events. Until now, it is easy to instantiate it but not to extend it. After that, we analyzed how to integrate it with DarX and how to extend the criticality analysis done by DarX.

In this section we will describe the proposed solution first from the XMLaw point of view, second from the DarX point of view and then the integration part.

6.1 XMLaw Extensions

The XMLaw conceptual model is composed by concepts which allow representing agents' interaction aspects. Among those concepts there are scenes, which group the interactions in modules; interaction protocols which define precisely the communication between agents; norms which hold the obligation, permission and prohibition concepts; clocks which add a temporal aspect into interactions allowing that a specific behavior may be valid during a period of time; and actions which allow executing automatic recovery code in the system in a fault case.

We have extended XMLaw with two new elements: *Role* and *Criticality Analysis* (Figure 10). XMLaw didn't have the concept of *Role*. The roles played by agents had to be informed by themselves when they ask to enter in an organization and it was associated to a new name of the agent inside the organization.

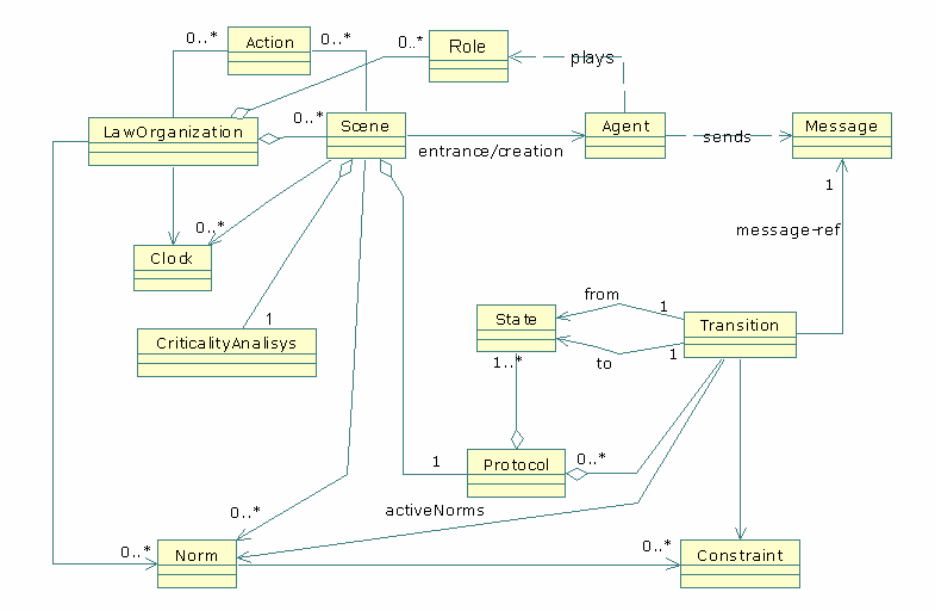

Figure 10 - XMLaw: New Conceptual Model

With this new element (*Role*), when an agent asks to enter in an organization, it has to inform the role it wants to play and when a scene is executed, the agent, if accepted, will have to play its role. An organization has one or more roles to be played by agents

and an agent can plays different roles in different organizations. Figure 11 shows the XML Schema which defines the legal building blocks of an XML. XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. XML Schemas are extensible to future additions, support data types and namespaces [29].

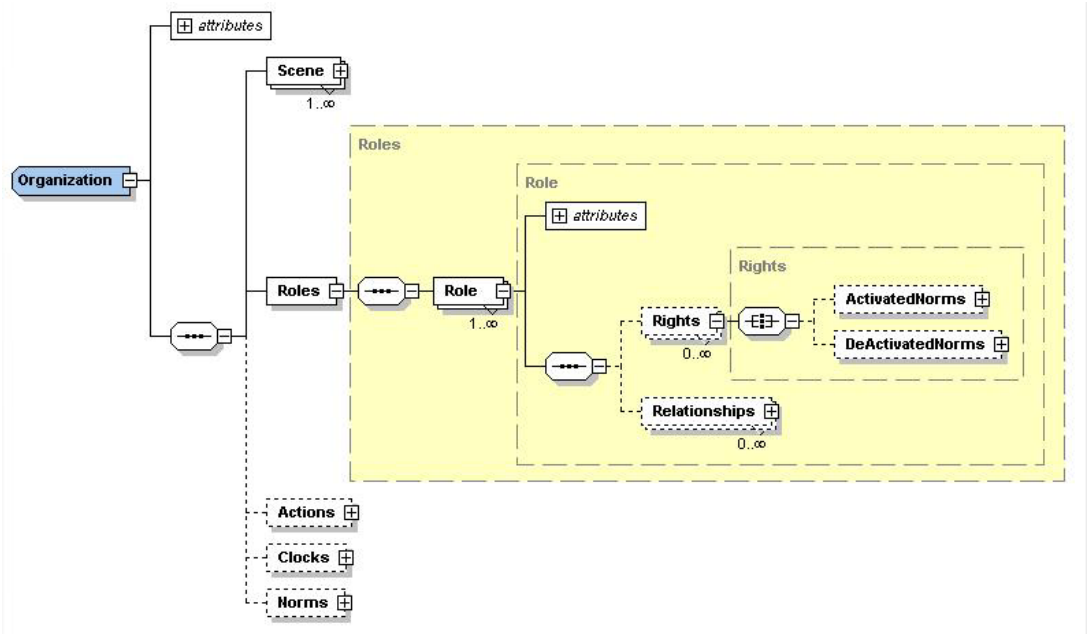

Figure 11 - XML Schema for Roles

The XMLaw specification is based on the XML Schema defined for it (Figure 12). The parser will read the law specification to generate the descriptors which are instantiated by the model level [27]. Then the mediator interacts with the model level in order to execute the law enforcement mechanism.

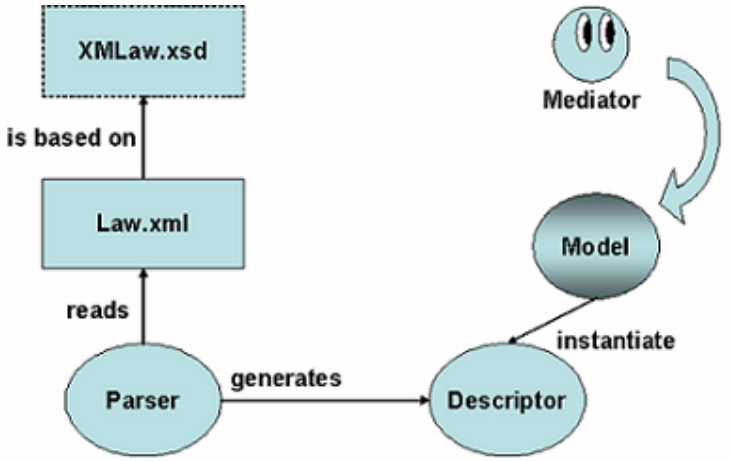

Figure 12 - XMLaw Framework execution

Each organization's role has identification and a list of norms associated to its *Rights* and its *Relationships*. Some norms can be activated and/or deactivated according to agents' right and agents' relationships. Rights describe the permissions on the resources and services available in the environment and about the behavior of the agents. It can activate or deactivate some norms related to that role. Relationships are defined as in [31]. Its definition is based on the protocols and commitments associated with the role. In this way, the agent role adds a set of relations to the agent that plays the role.

The criticality analysis element has two elements: *Increases* and *Decreases* (Figure 13). The element *Increases* has the list of events that contribute for the increase of the agent criticality. And the element *Decreases* has the list of events that contribute for the decrease of the agent criticality. Each element *Increase* and *Decrease* has as attributes: the event identification from the event that was fired, the event type from the event that was fired, the value which is a weight for the increase or decrease contribution of that event and the agent role identification which has all the references to the agent that will have its criticality updated in runtime.

For the problem described in the fourth section, for example, the XMLaw specification for the Criticality Analysis is showed in Figure 14. When an agent starts playing the customer role, its criticality has to be recalculated and updated by a weight of 0.3. The same happens when an agent starts playing the seller role, its criticality has to be updated by a weight of 0.7. Those actions are executed when the role activation event is fired.

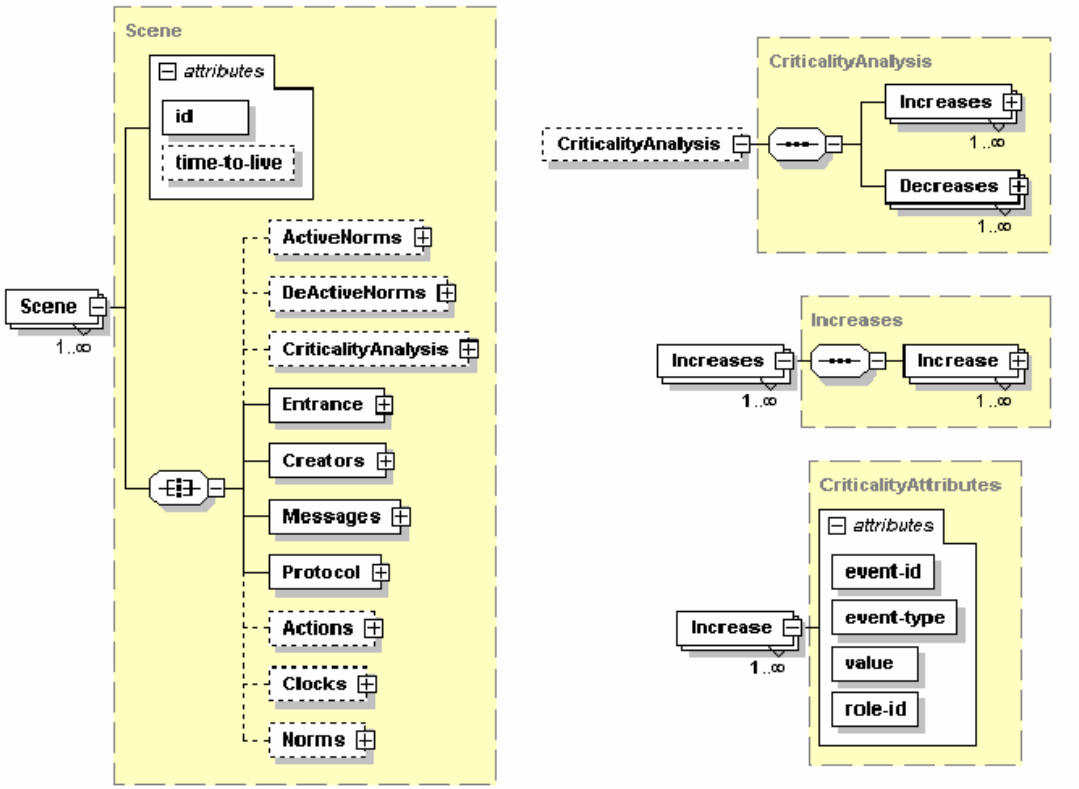

Figure 13 - XML Schema for Criticality Analysis

```
<CriticalityAnalysis>
    <Increases>
        <Increase event-id="customer" event-type="role_activation" value="0.3"
        role-id="customer" />
        <Increase event-id="seller" event-type="role_activation" value="0.7" role-
        id="seller" />
        <Increase event-id="time-to-decide" event-type="clock_activation"
        value="0.5" role-id="customer" />
        <Increase event-id="customer-payment-voucher" event-
        type="norm_activation" value="0.5" role-id="customer" />
    </Increases>
    <Decreases>
        <Decrease event-id="customer" event-type="role_deactivation"
        value="0.3" role-id="customer" />
        <Decrease event-id="seller" event-type="role_deactivation" value="0.7"
        role-id="seller" />
        <Decrease event-id="seller-permission-to-cancel" event-
        type="norm_activation" value="0.5" role-id="customer" />
    </Decreases>
</CriticalityAnalysis>
```

Figure 14 - XMLaw specification example for Criticality Analysis

6.2 DarX Extensions

In order to track the dynamical adoption of roles by agents, [12] proposed a role recognition method. Their approach is based on the observation of the agent execution and their interactions to recognize the roles of each agent and to evaluate its processing activity. This is used to dynamically compute the criticality of an agent through DarX (see third section).

They consider two cases. In the first case, each agent displays explicitly its roles or interaction protocols. The roles of each agent are thus easily deduced from its interaction events. In the second case, agents do not display their roles nor their interaction protocols. The agent roles are deduced from the interaction events by the role analysis module. In this analysis, attention is focused on the precise ordering of interaction events. The role analysis module captures and represents the set of interaction events resulting from the domain agent interactions (sent and received messages). These events are then used to determine the roles of the agent.

The analysis of events and measures (system data and interaction events) provides two kinds of information: the roles and the degree of activity of each agent. This information is then processed by the agent's criticality module. The latter relies on a table T that defines the weights of roles. This table is initialized by the application designer.

The criticality of the agent Agent *i* which fulfills the roles *ri1* to him is computed as follow:

$$wi(t) = (a1 * \text{aggregation}(T[rij]]j=1,m) + a2 * \text{awi}(t))/(a1 + a2)$$

Where *a1* and *a2* are the weights given to the two kinds of parameters (roles and degree of activity) which are introduced by the designer.

For each Agent *i*, its criticality *wi* is used to compute the number of its replicas. An agent is replicated according to:

– *wi*: its criticality,

– *W*: the sum of the domain agents' criticality,

– *rm*: the minimum number of replicas which is introduced by the designer,

– *Rm*: the available resources which define the maximum number of possible simultaneous replicas.

The number of replicas *nbi* of Agent *i,* which is used to update the number of replicas of the domain agent, can be determined as follows:

$$\boldsymbol{nbi(t)} = \text{rounded}(rm + wi(t) * Rm/W)$$

In our work we have proposed the same reasoning for updating the agents' criticality. For each value of increasing or decreasing agent's criticality it is stored on a table T which defines the weights of its event. So, for example, in our problem of negotiation scene there would be three different tables: *Tr,* which defines the weights of role activation or deactivation; *Tc,* which defines the weights of clocks activation or deactivation; and *Tn*, which defines the weights of norms activation or deactivation.

Then the criticality of the agent Agent *i* is computed as follow:

$$\boldsymbol{wi(t)} = (a1 * \text{aggregation}(Tr[rij]\ j=1,nr) + a2 * \text{aggregation}(Tc[cij]\ j=1,nc) + a3 * \text{aggregation}(Tn[nij]\ j=1,nn) + a4 * awi)) / \sum_{i=1}^{4} a_i$$

Where *a1, a2, a3* and *a4* are the weights given to the four kinds of parameters (roles, clocks, norms and degree of activity) which are introduced by the designer by XMLaw specification. And *awi* is the degree of activity of the agent *i*.

6.3 DarX & XMLaw Integration

Beyond the extensions made in XMLaw and DarX, it was necessary to create a bridge between DarX's agents and XMLaw's agents in order to integrate them. As you can see in Figure 15, a class AgentDarX was created in XMLaw framework. It extends DarXTask which allows distributed replication through DarX (see third section), and it has a reference to the Agent running in XMLaw.

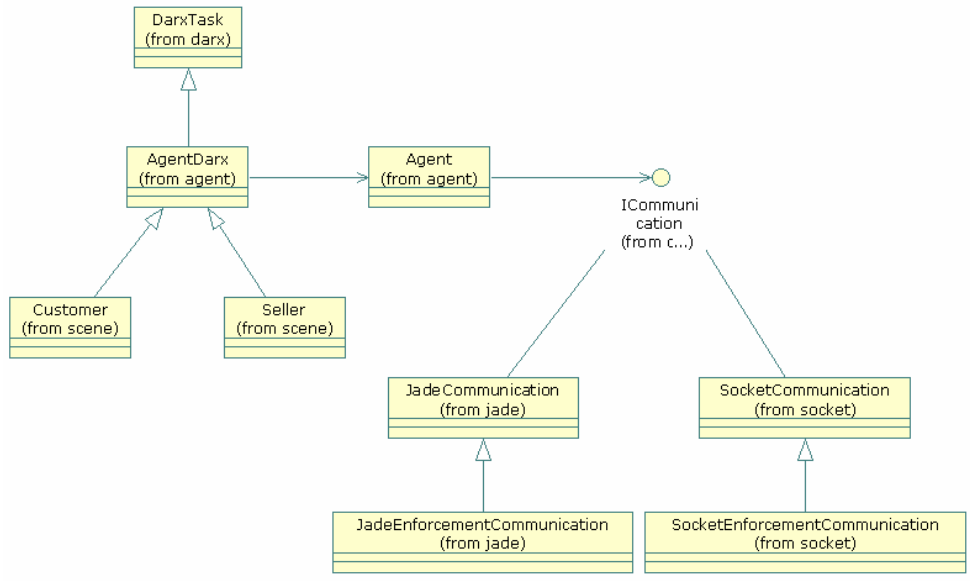

Figure 15 - Integration Class Diagram

Most of the methods implemented in Agent class were not copied to AgentDarX class. They were delegated to the reference of the agent, except the *sendSyncMessage*, *sendAsyncMessage*, *receiveSyncMessage* and *receiveAsyncMessage* which call the DarX strategies of replication mechanisms (active or passive) when synchronizing the messages through the replicas.

The two classes Customer and Seller in Figure 15 are there just for showing how to instantiate the new framework. All agents must extend from AgentDarX in order to beneficiate from replication strategies done by DarX and law-enforcement approach done by XMLaw.

XMLaw has its communication level extensible for other types of agent's communication. As it is not possible to serialize Jade [30], which is the platform used initially by XMLaw, it was created a simple communication level by socket. The socket just listens for new messages and provides a channel of sending it to the other agents.

7 Conclusions and Future Work

Massively multi-agents systems are often distributed and must run without any interruption. Moreover, considering that there are some kinds of systems that must be regulated by laws, the agents must be controlled by a law enforcement approach. To make these systems reliable, with a high degree of dependability where we can justifiably trust on it, we proposed a new approach to evaluate dynamically the criticality of agents during the law enforcement mechanism.

First it was necessary to extend XMLaw. Two new elements were introduced in the conceptual model: Role and Criticality Analysis. The first one (Role) was necessary because until now XMLaw hasn't these element and it would be very difficult to associate an event activation to the agent without having its reference. By doing that, we realized the needs of associating specific norms to an agent when it starts playing a role or stops playing it. Then we created the concept of Rights and Relationships. Rights describe the permissions on the resources and services available in the environment and about the behavior of the agents. It can activate or deactivate some norms related to that role.

Relationships are defined as in [31]. Its definition is based on the protocols and commitments associated with the role. In this way, the agent role adds a set of relations to the agent that plays the role.

The second element, Criticality Analysis, was introduced in order to monitoring the events that should improve the criticality analysis done by DarX. The events are divided in two groups: the ones that increase the agent's criticality and the ones that decrease it. By doing that, any event considered important by the designer of the application while specifying the law of it can be taken in account.

Second, it was necessary to extend DarX in order to provide another algorithm of calculating the agent's criticality. Considering the reasoning done by the Role Analysis described in [12], it was easy to extend it just instead of receiving one table with the weights, receiving tables related to XMLaw events with the weights.

During the XMLaw instantiation, we perceived that XMLaw could be improved in order to make it more extensible. First, the parser should perceive the Xml Schema and matches it with the XML, instead of just perceiving the XML. By doing that, it would be easy not only extend it but also verify the specification of the law designer as a way of guarantee that a wrong specification wouldn't be generated and executed by the mediator. Secondly, it is not appropriately documented, and it was hard to extend the parser and the model of execution. But once it was understood, it became trivial.

Another issue rose when we were wondering how we would specify a complex distributed system, as a multi-agents system by XMLaw. In our problem description, which was a negotiation scene, it wasn't necessary any strategy or methodology for the specification because it was too simple for that. But if we think in a bigger system with norms, scenes, protocols and maybe reuse of some scenes, we would like to have some kind of methodology or technique for doing it in the better way considering all the requirements of a system with high quality level, as maintainability, for example. How could we do it considering an increase on its dependability? Then, we are going to address those questions from now on.

8 Bibliography

[1] Xu, P., Deters, R., Fault-Management for Multi-Agent Systems.

[2] Carvalho, G., Paes, R., Choren, R., Lucena, C.. Governing the Interactions of An Agent-based Open Supply Chain Management System. MCC nº 29/05, Depto de Informática, PUC-Rio, 2005.

[3] Vpazquez-Salceda, J., Dignun, V., Dignun, F., Organizing Multiagent Systems, Autonomous Agentes and Multi-Agent Systems, 11, 307-360, 2005.

[4] F. Zambonelli, ''Abstractions and infrastructures for the design and development of mobile agent organizations,'' in M. Wooldridge, G. Weiss, and P. Ciancarini (eds.), Agent-Oriented Software Engineering II, LNCS 2222, Springer-Verlag, pp. 245–262, 2002.

[5] V. Dignum and F. Dignum, ''Modeling agent societies: Coordination frameworks and institutions,'' in P. Brazdil and A. Jorge (eds.), Progress in Artificial Intelligence, LNAI 2258, Springer-Verlag, pp. 191– 204, 2001.

[6] M. Esteva, J. Padget, and C. Sierra, ''Formalizing a language for institutions and norms,'' in J.-J. Ch. Meyer and M. Tambe (eds.), Intelligent Agents VIII, Vol. 2333 of LNAI, Springer-Verlag, pp. 348–366, 2001.

[7] F. Zambonelli, N. Jennings, and M. Wooldridge, ''Organisational abstractions for the analysis and design of multi-agent systems,'' in P. Ciancarini and M. Wooldridge (eds.), Agent-Oriented Software Engineering, LNCS 1957, Springer-Verlag, pp. 98–114, 2001.

[8] V. Dignum, J.-J. Ch. Meyer, H. Wiegand, and F. Dignum. ''An organisational-oriented model for agent societies,'' in G. Lindemann, D. Moldt, M. Paolucci, and B. Yu (eds.), Proceedings of the RASTA Workshop at AAMAS'02, pp. 31–50.

[9] R. Paes, G. R. Carvalho, C.J.P. Lucena, P. S. C. Alencar, H.O. Almeida; and V. T. Silva. Specifying Laws in Open Multi-Agent Systems. In: Agents, Norms and Institutions for Regulated Multi-agent Systems (ANIREM), AAMAS2005, 2005.

[10] Lussier, B. et al. 3rd IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments, Manchester (GB), 7-9 Septembre 2004, 7p.

[11] Peng Xu, Ralph Deters. "Using Event-Streams for Fault-Management in MAS," iat, pp. 433-436, IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04), 2004.

[12] Guessoum, Z., Briot, J.-P., Faci,N. Towards Fault-Tolerant Massively Multiagent Systems, Massively Multiagent Systems n.3446, LNAI, Springer Lecture Note Series, Verlag, 2005, pg. 55-69.

[13] Z. Guessoum et al. Dynamic and Adaptive Replication for Large-Scale Reliable Multi-Agent Systems. In Soft. Engineering for Large-Scale Multi-Agent Systems, LNCS 2603, pp. 182-198, April 2003.

[14] J. C. Laprie, J. Arlat, J. P. Blanquart, A. Costes, Y. Crouzert, Y. Deswarte, J. C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powel, C. Rabéjac, and P. Thévenod. Dependability Handbook (2nd edition) Cépaduès – Éditions, 1996. (ISBN 2-85428-341-4) (in French).

[15] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell: Dependability and its t-hreats - A taxonomy. IFIP Congress Topical Sessions 2004: 91-120.

[16] F. De Assis Silva and R. Popescu-Zeletin. An approach for providing mobile a-gent fault tolerance. In S. N. Maheshwari, editor, Second International Workshop on Mobile Agents, number 1477 in LNCS, pages 14–25. Springer Verlag, 1998.

[17] K. Decker, K. Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In ATAL'97, LNAI, pages 63–75. Springer Verlag, 1997.

[18] S. Hagg. A sentinel approach to fault handling in multi-agent systems. In C. Zhang and D. Lukose, editors, Multi-Agent Systems, Methodologies and Applications, number 1286 in LNCS, pages 190–195. Springer Verlag, 1997.

[19] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In AAMAS2002, Boulogna, Italy, 2002.

[20] R. Guerraoui, B. Garbinato, and K. Mazouni. Lessons from designing and implementing garf. In Proceedings Objects Oriented Parallel and Distributed Computations, volume LNCS 791, pages 238–256, Nottingham, 1989.

[21] M.Golm. Metaxa and the future of reflection. In OOPSLA -Workshop on Reflective Programming in C++ and Java, pages 238–256. Springer Verlag, 1998.

[22] Fedoruk, A. and Deters, R. 2003. Using dynamic proxy agent replicate groups to improve fault-tolerance in multi-agent systems. In Proc. of the Sec. int. Joint Conf. AAMAS '03. ACM Press, New York, NY, 990-991.

[23] Guessoum, Z., Faci, N., Briot, J.-P., Adaptive Replication of Large-Scale Multi-Agent Systems - Towards a Fault-Tolerant Multi-Agent Platform. Proc. of ICSE'05, 4th Int. Workshop on Soft. Eng. for Large-Scale Multi-Agent Systems, ACM Software Engineering Notes, 30(4) : 1-6, July 2005.

[24] R. Paes, C.J.P. Lucena and P.S.C. Alencar. A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems, 2005. http://www.les.inf.pucrio.br/governance/pubs.html

[25] R. Paes, G. Carvalho, C. Lucena, P. Alencar, H. Almeida, V. T. da Silva, Specifying laws in open multi-agent systems, in: Agents, Norms and Institutions for Regulated Multiagent Systems - ANIREM, Utrecht, The Netherlands, 2005.

[26] G. Cugola, E. D. Nitto, A. Fuggetta, Exploiting an event-based infrastructure to develop complex distributed systems, in: Proceedings of the 20th international conference on Software engineering, IEEE Computer Society, 1998, pp. 261–270.

[27] Paes, R., Alencar, P., Lucena, C. Governing Agent Interaction in Open Multi-Agent Systems. Monografias de Ciência da Computação nº 30/05, Departamento de Informática, PUC-Rio, Brazil, 2005.

[28] FIPA – The Foundation for Inteligent Physical Agents - Contract Net Interaction Protocol Specification http://www.fipa.org/specs/fipa00029/

[29] XML Schema. http://www.w3.org/XML/Schema, accessed in Nov, 21 2005.

[30] JADE – Java Agent DEvelopment Framework - http://jade.tilab.com/

[31] Silva, V. et.al: Taming Agents and Objects in Software Engineering, In Software Engineering for Large-Scale Multi-Agent System, Garcia, A. et. al Eds., LNCS, Springer, 2003.

[32] Carl Hewitt and Peter de Jong. Open systems on conceptual modelling. Technical report, Massachusetts Institute of Technology, 1982.

[33] Jennings, Nicholas R., An Agent-Based Approach for Building Complex Software Systems, Communications of the ACM, 44(4), 35-41, April 2001.

[34] Jennings, N. R., "Agent-oriented software engineering", Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI, (1999).

[35] Paes, R.B; Lucena, C.J.P; Alencar, P.S.C. A Mechanism for Governing Agent Interaction in Open Multi-Agent Systems MCC nº 30/05, Depto de Informática, PUC-Rio, 31 p., 2005.

[36] Lussier, B. et al. 3rd IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments, Manchester (GB), 7-9 Septembre 2004, 7p.

[37] Vpazquez-Salceda, J., Dignun, V., Dignun, F., Organizing Multiagent Systems, Autonomous Agentes and Multi-Agent Systems, 11, 307-360, 2005.

[38] Weinstock, C.B., Goodenough, J.B., Hudak, J.J., Dependability Cases, Technical Note, CMU/SEI-2004-TN-016, 2004.